Vacuum World Reflex Agent

This assignment was ported from content at: https://notebook.community/jo-tez/aima-python/vacuum_world

**STUDENT NAME: **

STEP 1

Go to File -> Save copy in Drive

You shouldn't try to edit the read only notebook

A simple reflex agent program selects actions on the basis of the current percept, ignoring the rest of the percept history. These agents work on a condition-action rule (also called situation-action rule, production or if-then rule), which tells the agent the action to trigger when a particular situation is encountered.

I have a lot of imported code. The sections should be collapsed. Much of it is not needed. I copied entire class files from the source without editing. You can look over the code, but I recommend just ignoring it.

```
# These libraries do not come built in with collab
# you need run the imports once
# Each time the environment is destroyed you run it again

# Uncomment the next two lines if you need to install ipythonblocks and qpsolvers
#!pip install ipythonblocks
#!pip install qpsolvers
```

You need to run all the cells in this section but do not need to change anything

Imports

Show code

Utils.py

Show code

Agents.py

Show code

Probabilistic Learning

Show code

Search.py

Show code

CSP.py

Show code

Logic.py

Show code

Learning.py

Show code

Now we are getting to the real program

We will start by walking through a trivial environment. I will provide code for a random agent in the trivial environment

```
class TrivialVacuumEnvironment(Environment):

    """This environment has two locations, A and B. Each can be Dirty
    or Clean. The agent perceives its location and the location's
    status. This serves as an example of how to implement a simple
    Environment."""

    def __init__(self):
        super().__init__()
        self.status = {loc_A: random.choice(['Clean', 'Dirty']),
                       loc_B: random.choice(['Clean', 'Dirty'])}

    def thing_classes(self):
        return [Wall, Dirt, ReflexVacuumAgent, RandomVacuumAgent,
                TableDrivenVacuumAgent, ModelBasedVacuumAgent]

    def percept(self, agent):
        """Returns the agent's location, and the location status (Dirty/Clean)."""
        return (agent.location, self.status[agent.location])
```

```
    def execute_action(self, agent, action):
        """Change agent's location and/or location's status; track performance.
        Score 10 for each dirt cleaned; -1 for each move."""
        if action == 'Right':
            agent.location = loc_B
            agent.performance -= 1
        elif action == 'Left':
            agent.location = loc_A
            agent.performance -= 1
        elif action == 'Suck':
            if self.status[agent.location] == 'Dirty':
                agent.performance += 10
            self.status[agent.location] = 'Clean'

    def default_location(self, thing):
        """Agents start in either location at random."""
        return random.choice([loc_A, loc_B])
```

```
# These are the two locations for the two-state environment
loc_A, loc_B = (0, 0), (1, 0)

# Initialize the two-state environment
trivial_vacuum_env = TrivialVacuumEnvironment()

# Check the initial state of the environment
print("State of the Environment: {}.".format(trivial_vacuum_env.status))
```

```
# Create the random agent
random_agent = Agent(program=RandomAgentProgram(['Right', 'Left', 'Suck', 'NoOp']))
```

```
# Add agent to the environment
trivial_vacuum_env.add_thing(random_agent)

print("RandomVacuumAgent is located at {}.".format(random_agent.location))
```

```
# Running the environment
trivial_vacuum_env.step()

# Check the current state of the environment
print("State of the Environment: {}.".format(trivial_vacuum_env.status))

print("RandomVacuumAgent is located at {}.".format(random_agent.location))
```

```
# Now lets move again and see what happens
# After everything is clean, you will need to reset the environment
trivial_vacuum_env.step()

# Check the current state of the environment
print("State of the Environment: {}.".format(trivial_vacuum_env.status))

print("RandomVacuumAgent is located at {}.".format(random_agent.location))
```

```
trivial_vacuum_env.delete_thing(random_agent)
```

Reflex Agent in Trivial Environment

```
# This is example code for a two location reflex agent
loc_A = (0, 0)
loc_B = (1, 0)

"""We change the simpleReflexAgentProgram so that it doesn't make use of the Rule class"""
def SimpleReflexAgentProgram():
    """This agent takes action based solely on the percept. [Figure 2.10]"""

    def program(percept):
        loc, status = percept
        return ('Suck' if status == 'Dirty'
                else'Right' if loc == loc_A
                            else'Left')
    return program


# Create a simple reflex agent the two-state environment
trivial_vacuum_env = None # make sure this is clear
trivial_vacuum_env = TrivialVacuumEnvironment()
program = SimpleReflexAgentProgram()
simple_reflex_agent = Agent(program)
```

```
trivial_vacuum_env.add_thing(simple_reflex_agent)
```

```
# Check the current state of the environment
print("State of the Environment: {}.".format(trivial_vacuum_env.status))
print("SimpleReflexVacuumAgent is located at {}.".format(simple_reflex_agent.location))
```

```
# Run the environment
trivial_vacuum_env.step()

print("State of the Environment: {}.".format(trivial_vacuum_env.status))
print("SimpleReflexVacuumAgent is located at {}.".format(simple_reflex_agent.location))
```

```
# We can loop until we have a clean / clean state but I just want to show general concept
```

Now you need to update the reflex agent to handle a 2x2 enviroment

- Finish the TwoByTwoVacuumEnvironment to handle a 2x2 environment
- If you are at a dirty location suck up the dirt
- If the location is clean, move clockwise
    - To define clockwise, assume (0,0) is top left and (1,1) is bottom right

- The end state should be that all squares are clean
- We can give the vacuum enough intelligence to not loop infinitely

```
# 2x2 Vacuum world locs
loc_A1, loc_A2, loc_B1, loc_B2 = (0, 0), (0, 1), (1,0) ,(1,1)

class TwoByTwoVacuumEnvironment(Environment):

    """This environment has two locations, A and B. Each can be Dirty
    or Clean. The agent perceives its location and the location's
    status. This serves as an example of how to implement a simple
    Environment."""

    def __init__(self):
        super().__init__()
        self.status = {loc_A1: random.choice(['Clean', 'Dirty']),
                       loc_A2: random.choice(['Clean', 'Dirty']),
                       loc_B1: random.choice(['Clean', 'Dirty']),
                       loc_B2: random.choice(['Clean', 'Dirty'])
                       }

    def thing_classes(self):
        return [Wall, Dirt, ReflexVacuumAgent, RandomVacuumAgent,
                TableDrivenVacuumAgent, ModelBasedVacuumAgent]

    def percept(self, agent):
        """Returns the agent's location, and the location status (Dirty/Clean)."""
        return (agent.location, self.status[agent.location])

    def execute_action(self, agent, action):
        ### STUDENT TO DO: UPDATE THIS METHOD TO HANDLE 2x2
        pass

    def default_location(self, thing):
        """Agents start in either location at random."""
        return random.choice([loc_A1, loc_A2,loc_B1,loc_B2])
```

```
# Finish this function to use similar logic as the trivial example
def TwobyTwoReflexAgentProgram():
    """This agent takes action based solely on the percept. """

    def program(percept):
        # Use code from simple environment for inspiration to do this one
        pass



    return program
```

```
def clean_all():
  # Write a function that will instantiate the environment and clean all the squares
  # The test script will call this function and validate the environment you return
  # Make sure to avoid infinite loops
```

```
  box_environment = None # Just a placeholder

  # Your code

  return box_environment
```

```
box = clean_all()

box.status
```

▸ Grading Script

Show code